Executable Aspect Oriented Models for Improved Model Testing

Andrew Jackson, Jacques Klein, Benoit Baudry, Siobhan Clarke

Executable Aspect Oriented Models for Improved Model Testing

Andrew Jackson[1], Jacques Klein[2], Benoit Baudry[2], Siobhán Clarke[1]

[1] Distributed Systems Group, Computer Science Dept., Trinity College Dublin, Dublin, Ireland.
{Andrew.Jackson@cs.tcd.ie}
[2] IRISA, Campus Beaulieu, 35042 Rennes Cedex, Rennes, France.
{jacques.klein, benoit.baudry}@irisa.fr

Abstract. Design validation is important for detecting errors early in the development life cycle. Testing the design is one significant means to achieve design validation. In this paper we introduce the KerTheme model. KerTheme provides a means symmetrically decomposing concern based executable class diagrams and concern test scenarios. KerTheme also facilitates simultaneous merging of these decomposed models into a coherent composite concern based executable class model and corresponding test scenarios. The KerTheme model allows us to investigate weather decomposed concern based executable class diagrams simplifies the definition of concern test scenarios. This will also allow us to investigate weather this approach ensures more rigorous testing of a complete system.

1 Introduction

Design validation is important for detecting errors early in the development life cycle. The earlier an error is detected, the easier and cheaper it is to resolve. Testing the design is one significant means to achieve design validation [6].

There are various ways to design a system. One popular means to mitigate errors is to employ test driven development. Typically, tests are formulated based on use cases which often represent concerns. The concern decomposition facilitated by use case modelling is generally lost when using traditional design paradigms (e.g. object-orientation). This is because these paradigms do support concern modules as a first class construct in design.

When concerns are scattered and tangled in one monolithic design, the design becomes harder to test because it is harder to write a test case that targets one concern in complete isolation. If concerns are scattered and tangled an error in the design of one concern can have a negative impact on other concerns with which it is entangled. As such, it is difficult to detect the error and localize the effect of resolution.

The UML enables the designer to separate some kinds of concerns. Aspect oriented modelling (AOM) approaches typically extends the UML increasing the scope for concern separation at design time. It is claimed that concern separation improves

design reusability, compensability and flexibility [5]. We are investigating the extent to which concern separation at the design level also improves the testability of design. Design models that represent concerns are focused on one area of interest in a system. Through this work we are investigating whether it is easier to express scenarios as test cases for a specific concern. Moreover, we expect that when an error is detected; it will be easier to identify the precise causes of errors and resolve them quickly.

Various AOM approaches exist for separating concerns within the design space [7]. Theme/UML [4] is unique within this space as it provides both a symmetric decomposition model and well defined composition semantics. A symmetric decomposition model ensures that both crosscutting and non-crosscutting concerns can be modularized. In Theme/UML crosscutting and non-crosscutting concerns are modularized as themes. Themes encapsulate the standard UML structural and behavioural diagrams required to capture the concerns structure and behaviour. Well defined composition semantics describe the effect the composition operator (e.g., merge) has on themes.

Testing is checking the consistency between what the developer wants and what the designer has. Typically this would be the execution of a test case against a program. In our case, we need to be capable of comparing two views on the same design model (what the developer wants and what the designer has). Theme/UML provides two views that would allow us to test a Theme model (behavioural and structural diagrams). Behavioural diagrams have been illustrated as a good means for generating test cases [6]. They define some particular expected traces through a system based on a defined context. The behavioural diagrams for a particular theme can be used to describe test cases. For a theme to be testable, the structural diagrams need to be executable.

In our initial work we are augmenting the Theme/UML model to be executable. We have implemented a prototype Theme/UML model on the Kermeta platform. The Kermeta platform enables the weaving of executability into object-oriented metalanguages. Theme/UML extends the UML by defining the composition of UML models, modularized in themes. By implementing the Theme/UML model in Kermeta we are able to weave executability into themes and define the composition of executable themes.

Kermeta enables executable behaviour to be woven into structural models [9]. In Theme/UML behaviour is typically described through behavioural diagrams. When themes are executable, diagrammatic representation of behaviour is no longer necessary as the behaviour may be observed through model execution. In existing work, the weaving of scenario models has been investigated and formally specified [8]. Scenario composition is also being implemented on the Kermeta platform.

In this paper we propose the modularization of test scenarios and executable theme models within a new KerTheme model. Like the original Theme/UML model the KerTheme model provides two perspectives the executable theme model describes what the designer has and the accompanying scenarios describe what the designer wants.

It is our intuition that we will be able to test theme designs by making themes executable and defining test scenarios for these executable themes. By comparing scenarios and execution paths through theme models it may be possible to validate a

theme in isolation. However, in most cases themes need to be composed to ensure correct execution. In this paper we outline a means for a consistent merge of executable class diagrams and sequence diagrams to generate both a composite test case and composite theme model, whereby the composite test case fully validates the composite theme model.

In section 2 we describe the background to this work: Section 2.1 introduces Theme/UML; Section 2.2 describes Kermeta; and Section 2.3 presents Semantic based Scenario Weaving. Section 3 describes our approach using an example derived from an Auction System case study. Section 4 outlines the benefits and limitations of the approach and also includes a brief discussion. We present related work in Section 5 and conclude the paper, outlining future work in Section 6.

2 Background

In this paper we motivate and illustrate the benefits of integration through an Auction System Case study[1]. This case study has previously been used in [7, 8]. In this paper we focus on two concerns, a login concern and a persistence concern, which are part of the Auction System. The persistence concern crosscuts the login concern at points in the execution of the login concern where login attempts are made against the system. The persistence concern deals with the recording of login attempts.

2.1 Theme/UML

Theme/UML [4] is a MOF based extension [5] to the UML that supports AOM. Theme/UML facilitates the symmetric concern based decomposition of a system and the specification of (base and aspect) design modules are to be composed. In Theme/UML composition is specified as a merge relationship between themes. The top half of Figure 1 illustrates the design of the login and persistence concerns as Themes. The login theme is a *base theme* and the persistence is an *aspect theme*. A *base theme* is an extension of the UML package meta-class, instances of which encapsulate the structural and behavioural UML diagrams that the designer needs to describe a concern. An *aspect theme* extends the template package meta-class. The *template paramaters* associated with the template package are used in the description of *crosscutting behavior*. The *template paramaters* are representative of any join points at which the *aspectual behaviour* affects. In the login and persistence themes at the top of Figure 1, class and sequnce diagrams are used to describe the structure and behaviour of concerns.

Figure 1 also illustrates the specification of a *merge* composition of the persistence and login concerns. As persistence is crosscutting the merge specification a *join point binging* is specified as part of the *merge*. A join point binding specifies elements that exist within themes as points that are to be crosscut. In Figure 1 the

[1] A description of the Auction System is available at http://lgl.epfl.ch/research/fondue/case-studies/auction/problem-description.html

`Server.login(): Boolean` method is bound to the template paramater exposed by the persistence theme. The result if this binding is presented at the bottom of Figure 1 in the Auction *composite theme*. The class diagrams that were defined in the persistence and login themes are unified into one class diagram. There are two sequence diagrams in the composite theme. The login sequence diagram (SD) is equivalent to the login SD in the login theme. The persist SD shows how the join point specified in the *join point binding* replaces the *template parameter* to describe composed behaviour.

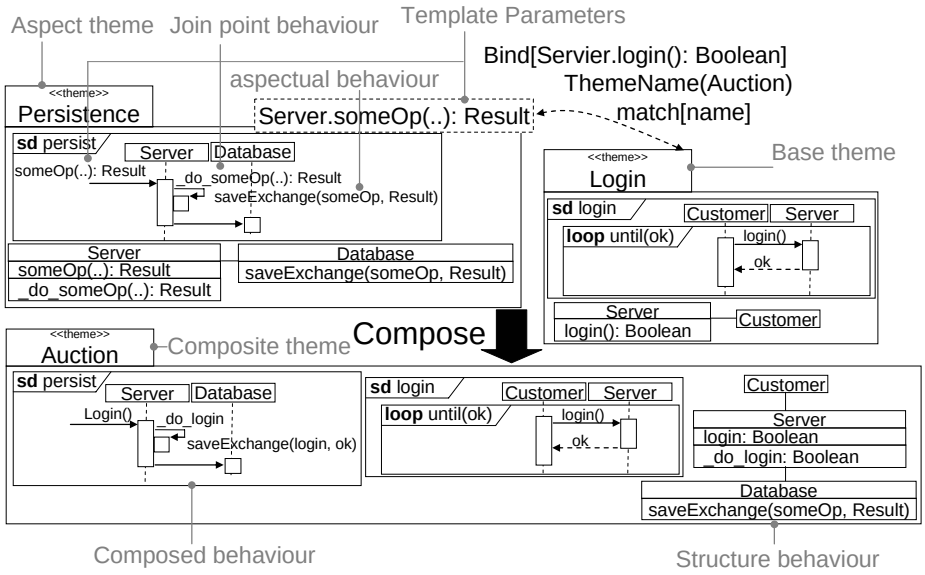

Figure 1 Composition of login and persistence themes

2.2 Kermeta

Kermeta [8] is a meta-modeling language that has been designed as an extension to the EMOF 2.0 to be the core of a meta-modeling platform. Kermeta extends EMOF with an action language that allows the specification of behavioral semantics formetamodels. This action language is imperative and object-oriented. It is used to provide an implementation of operations defined in metamodels. As a result the Kermeta language can, not only be used for the definition of metamodels but also for implementing their semantics, constraints and transformations.

The Kermeta action language has been specially designed to process models. It includes both Object Oriented (OO) features and model specific features. Kermeta includes traditional OO static typing, multiple inheritance and behavior redefinition/selection with a late binding semantics. To make Kermeta suitable for model processing, more specific concepts such as opposite properties (i.e. associations) and handling of object containment have been included. In addition to this, convenient

Object Constraint Language (OCL) constructs, such as closures (e.g. each, collect, select), are also available in Kermeta.

A complete description of the language definition can be found in [8]. Kermeta has been used in the successful implementation of class diagram composition in [15] but also as a model transformation language in [10]. To implement the detection and composition techniques proposed in this paper we have chosen to use Kermeta for two reasons. First, the language allows implementing composition by adding the algorithm in the body of the operations defined in the composition metamodel. Second, Kermeta tools are compatible with the Eclipse Modeling Framework (EMF) [3]which allows us to use Eclipse tools to edit, store, and visualize models.

2.3 Semantic Based Scenario Weaving

In [8], Klein et al. propose a semantic-based weaving of scenarios, where the weaving is based on the dynamic semantics of the models used. In particular, it takes into account all the behaviors that a Sequence Diagram (SD) can define, even if the number of these behaviors is infinite. In this approach, an aspect is described as behavioral aspect because it is specified with behavioral modeling languages. Indeed, an aspect is defined as a pair of SDs, one SD for the pointcut (specification of the behavior to detect), and the second one for an advice representing the expected behavior at the join point. Similarly to Aspect-J, where an aspect can be inserted 'around', 'before' or 'after' a join point, with the approach defines in [8], an advice may indifferently complete the matched behavior, replace it with a new behavior, or remove it entirely.

One of the difficulties to weave SDs is that we have to weave a dynamic behavior at modeling time. Therefore, we need to statically find where in the base scenarios are the join points. While this can be trivially implemented with a syntactic match for simple SDs, the hierarchical nature of UML 2.0 SD (similar to HMSCs [16]) makes it necessary to address the problem at the semantic level [8] with static analysis techniques such as loop unrolling, etc.

3 Merging Models and Tests

To facilitate our investigation into the extent to which concern separation at the design level also improves the testability of design, we are combining the Theme/UML and the semantic based weaving scenarios on the Kermeta platform to create a new KerTheme model.

The symmetric decomposition model and well defined composition model defined by Theme/UML is extended on the Kermeta platform to define executable class models. As we elucidated in Section 1, testing is checking the consistency between what the developer wants and what the designer has. We expect that symmetric concern decompositions will make it easier for the developer to describe what he/she wants in from a specific concern.

Once developed the executable class model is what the designer has. For defining what the designer wants, we use scenarios. Scenarios allow the designer to concisely define expected execution paths through the executable based on an initial state. Scenarios are defined in sequence diagrams that can contain control structures such as loops and conditionals. Testing is then a matter of checking the consistency between a resulting state and the scenario defined for an executable class model a resulting execution trace based on an initial state.

Although we hypothesize concern design and concern based test development makes testing of models easier, we relaise that validating a system design is the overall goal. The KerTheme model facilitates the synchronized composition of both concern model and concern test scenarios. A merge operator that is based on an integration of the PackageMerge and Theme/UML semantics is defined for the composition of executable class models. Semantic based scenario weaving is defined for the composition of concern test scenarios.

Figure 2 illustrates an example of the KerTheme model in which there are two executable KerThemes LogIn and Persistence. The «KerTheme» stereotype denotes themes as executable and testable. Each KerTheme contains both executable classes (executability is represented by a lightening symbol) and sequence diagrams which represent test case scenarios. An executable class is one where by execution logic (expressed in the Kermeta Language) is woven into the methods declared on the meta-class.

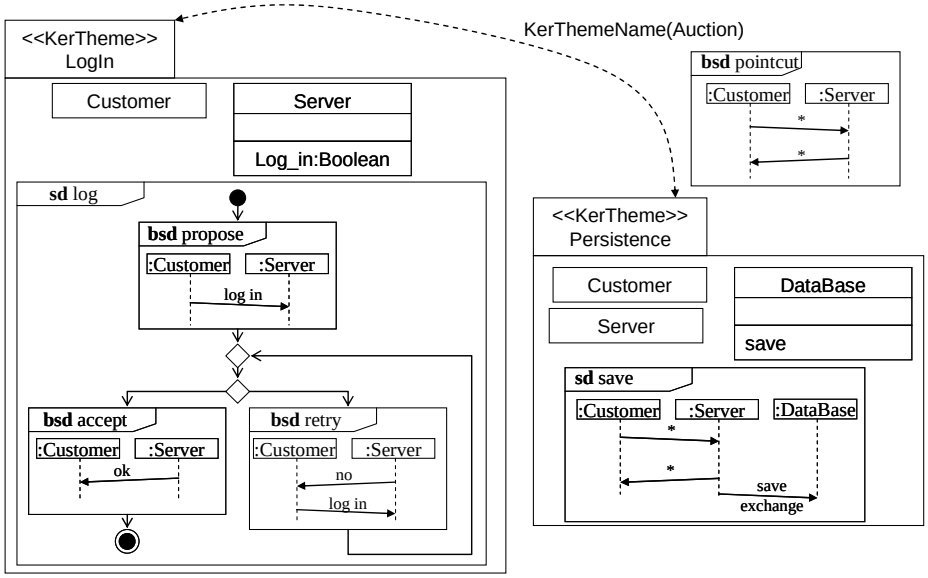

Figure 2 KerTheme models for LogIn and Persistence concerns

The LogIn KerTheme is testable because the logic encapsulated in the executable classes models and the resulting execution path and result can be compared against the sequence diagram test cases. This is not the case for the Persistence KerTheme, as

both KerTheme class model and sequence diagram test cases are parameterized and incomplete until bound through composition.

To test the persistence behaviour we need to merge both the executable class model and the sequence diagrams test cases with the LogIn executable class model and sequence diagrams test cases. Figure 2 illustrates the composition of the LogIn and Persistence KerThemes. In this diagram the executable class models are unified and the sequence diagrams are composed. Through comparing the execution path of this composed model against the sequence diagram test case we can ensure that the logic that is described in the persistence KerTheme is error free. We can also regression test the composite Auction KerTheme with the sequence diagram test case defined in LogIn KerTheme.

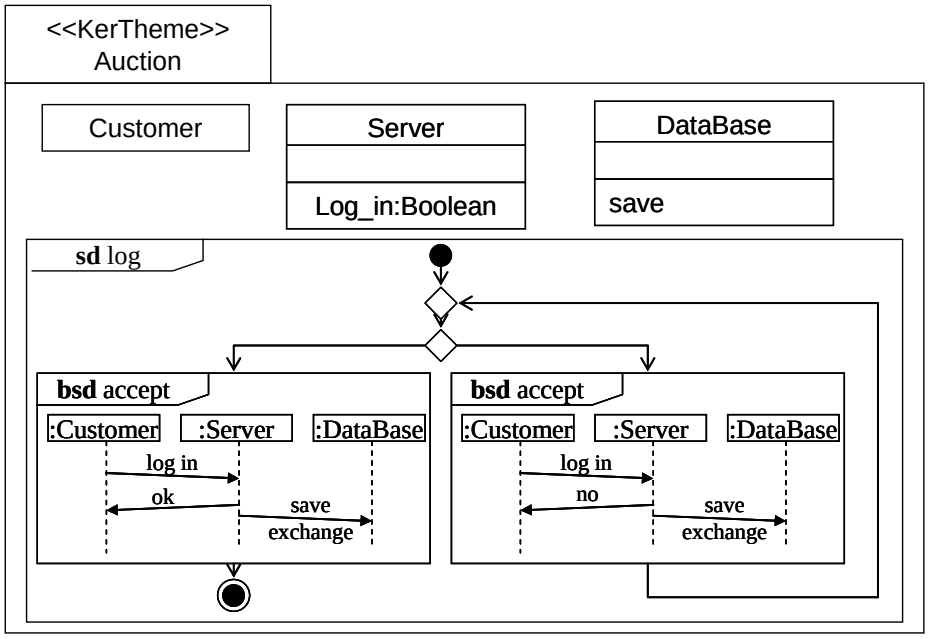

Figure 3 Composite KerTheme design model

4 Discussion, Benefits and Limitations

The benefits of this approach include:

1. **The correctness of the design model can be validated**: the designer can be more confident that errors in the design logic will not emerge during implementation.
2. **Test cases are easier to create and change**: the designer can focus on creating tests for one concern in isolation and as such tests are more concentrated and easier to change.

3. **Finding errors is easier**: when a test fails it is possible to relate the error to a particular concern. As we check execution paths against sequence diagram test cases, it should be easy to identify where errors arise.
4. **The affect of fixing errors in minimised**: Changes to the design model are localized within a KerTheme and as such the negative effects of change are reduced.
5. **Improved Reusability**: As KerTheme represents a modular unit of design logic and tests for that design logic, KerThemes are easier to reuse.

We have also identified limitations of this approach. These include:

1. **May need global tests**: it may be difficult to write test cases in isolation in some cases it is necessary to add global test cases.
2. **Weaving correctness**: It may be difficult to ensure that tests and models have been composed correctly, unless additional tests are applied to the composite.

Let us note that an executable class diagram is not the same than an object-oriented program. Firstly, with an executable class diagram, the level of abstraction is higher: it is independent of a platform.

Secondly, when a class diagram is implemented with an OO language, the initial model is not really preserved. For instance, the associations between classes are changed into attributes, or multiple inheritances are removed by using interface, etc… These differences between model and OO code make that it is difficult to continue to properly work at a model level as soon as an OO language is used, whereas with an executable class diagram, where the code is added in the methods, the static model remains unchanged: so it is easier to continue to work at a model level.

5 Related Work

A number of works study the use of sequence diagrams to define and generate test cases. In [12], the authors propose a technique to automatically generate test cases for UML design models. The UML design models consist in a class diagram, OCL pre and post conditions for methods and activity diagrams that model the behaviour of each method. From this design model, the authors can generate an executable form of the model which can be tested using dynamic testing techniques. Concerning test generation, they propose to model test cases using UML2.0 sequence diagrams. From these test cases specification and the class diagram, they generate a graph that corresponds to all possible execution paths defined in the different scenarios. The authors then use coverage criteria defined in [14]. From the graph, it is possible to automatically generate test data and an initial system configuration to cover each execution path.

Other works also propose to use scenarios as a basis to generate cases for programs. In [11, 12], Pickin et al. investigate the use of sequence diagrams as a formal language to write test cases for distributed systems. The UML model of the system has to be composed of a class diagram and a state diagram. Then, from test objectives, modelled with sequence diagrams and a description of the initial state of the

application in the form of an object diagram, the authors propose a technique to synthesize test cases for the system. The approach uses input/output labelled transition systems (IOLTS) as an intermediate formal model from which to generate test cases. Thus, they have a transformation [6] from a UML model to IOLTS, and a reverse transformation to represent the generated test cases with sequence diagrams. The test cases specify the ordering of call sequences that should be sent to the system and associated test verdicts.

The Object Management Group also considers sequence diagrams as a practical language to write test cases for object-oriented systems. The UML testing profile [14] proposes a metamodel for that captures all the concepts needed to design and generate black-box test cases: test objective, test case, test data, test environment, system under test (SUT)... In this profile, the behaviour of a test case (the behaviour a test case aims at validating) is described using a sequence diagram. The profile also proposes mappings from test scenarios to JUnit or TTCN.

In [2] Briand et al. propose to use scenarios associated to use cases as high level test cases. The authors propose to use activity diagrams to model the ordering of use cases. It is then possible to extract paths from the activity diagram, which correspond to legal sequences of use cases. Once these sequences are available, each use case is replaced by its corresponding sequence diagram to build a global test case. The generated test cases correspond to requirements level test cases that should be executed on the final implementation of the system.

6 Conclusions and future work

Although work is being done in the area of testing in AOSD [17], little of this work is focusing on the model level. An innovation of our approach is to take a well defined symmetric AOM model and enhance this model with means to test individual concerns through the definition of simple tests. Another innovation of this work is the definition of a merge operator that synchronizes the merging of both models and the tests with which correspond to these tests. By composing tests we can execute tests and models and test composite tests against composite models and evaluate the result. Where the results are negative there may be problems with the composition as specified.

In our future work we will further investigate and extend the proposition presented in this paper. We will continue to investigate other means for testing AO based models at design time. We will also evaluate our approach through the Auction Case study described and used in [7] and [8].

References

[1] Andrews, A., France, R., Ghosh, S., Craig, G.: Test adequacy criteria for UML design models. Software Testing, Verification and Reliability, 2003. 13(2): 95 -127. [2]

[2] Briand, L., Labiche, Y.: A UML-based approach to System Testing. Software and Systems Modeling, 2002. 1(1): 10 – 42 [6]

[3] Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose T., Eclipse Modeling Framework. The Eclipse Series. 2003: Addison Wesley Professional. [9]

[4] Clarke, S., Baniassad, E.: Aspect-Oriented Analysis and Design the Theme Approach, ISBN: 0321246748, 2005 Addison-Wesley

[5] Clarke, S., Walker, R. J.: Composition Patterns: An Approach to Designing Reusable Aspects, 23rd International Conference on Software Engineering (ICSE), Toronto, Canada, May 2001.

[6] Dinh-Trong, T., Kawane, N., Ghosh, S., France, R., Andrews A. A.: A Tool-Supported Approach to Testing UML Design Models, 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), June 2005, Shanghai, China [1]

[7] Jackson, A., Clarke, S., Initial Version of Aspect-Oriented Design Approach, aosd-europe.net, Feburary, 2006

[8] Klein, J., Helouet L., Jézéquel, J.M.: Semantic-based Weaving of Scenarios, 5th International Conference on Aspect-Oriented Software Development (AOSD' 06), March 2006. Bonn, Germany [11]

[9] Muller, P.A., Fleurey, F., Jézéquel, J.M.: Weaving executability into object-oriented meta-languages, Proceedings of MODELS/UML'2005, volume 3713 of LNCS, pages 264-278, October 2005, Montego Bay, Jamaica [10]

[10] Muller, P.A., Fleurey, F., Vojtisek, D., Drey, Z., Pollet, D., Fondement, F., Studer, P., Jezequel, J.M.: On Executable Meta-Languages applied to Model Transformations, in Model Transformations In Practice Workshop, 2005. Montego Bay, Jamaica.

[11] Pickin, S., Jézéquel J.M.: Using UML sequence diagrams as basis for a formal test description language. Proceedings of IFM2004 (Fourth International Conference on Integrated Formal Methods), Canterbury, Kent, England, 2004. [3]

[12] Pickin, S., Jard, C., Le Traon, Y., Jéron, T., Jézéquel J.M., Le Guennec, A.: System Test Synthesis from UML Models of Distributed Software. Proceedings of FORTE, 2002. [4]

[13] UML Superspec p107-115, http://www.omg.org/, 2004.

[14] UML Testing Profile, Accessed on: April 2006. http://www.omg.org/docs/formal/05-07-07.pdf [5]

[15] Reddy, R., France, R., Ghosh, S., Fleurey, F., Baudry, B.: Model Composition - A Signature-Based Approach. Aspect Oriented Modeling (AOM) Workshop. 2005. Montego Bay, Jamaica.[8]

[16] TS, I., ITU-TS Recommendation Z.120: Message Sequence Chart (MSC). 1993, Geneva: ITU-TS [12]

[17] Xu, D., Xu, W.: State-Based Incremental Testing of Aspect-Oriented Programs, 5th International Conference on Aspect-Oriented Software Development (AOSD' 06), March 2006. Bonn, Germany